# Designing A General Deep Web Access Approach Based On A Newly Introduced Factor; Harvestability Factor (HF)

Mohamamdreza Khelghati, Maurice van Keulen, Djoerd Hiemstra

Databases Group, University of Twente, Netherlands
s.m.khelghati, m.vankeulen, d.hiemstra@utwente.nl

**Abstract.** The growing need of accessing more and more information draws attentions to huge amount of data hidden behind web forms defined as deep web. To make this data accessible, harvesters have a crucial role. Targeting different domains and websites enhances the need to have a general-purpose harvester which can be applied to different settings and situations. To develop such a harvester, a number of issues should be considered. Among these issues, business domain features, targeted websites' features, and the harvesting goals are the most influential ones. To consider all these elements in one big picture, a new concept, called harvestability factor (HF), is introduced in this paper. The HF is defined as an attribute of a website ($HF_W$) or a harvester ($HF_H$) representing the extent to which the website can be harvested or the harvester can harvest. The comprising elements of these factors are different websites' (for $HF_W$) or harvesters' (for $HF_H$) features. These features are presented in this paper by gathering a number of them from literature and introducing new ones through the authors' experiments. In addition to enabling websites' or harvesters' designers of evaluating where they products stand from the harvesting perspective, the HF can act as a framework for designing general purpose deep web harvesters. This framework allows filling in the gap in designing general purpose harvesters by focusing on detailed features of deep websites which have effects on harvesting processes. The represented features in this paper provide a thorough list of requirements for designing deep web harvesters which is not done to best of our knowledge in literature in this extent. To validate the effectiveness of HF in practice, it is shown how the HFs' elements can be applied in categorizing deep websites and how this is useful in designing a harvester. To run the experiments, the developed harvester by the authors, is also discussed in this paper.

## 1 Introduction

Nowadays, in an information-thirsty environment, the *deep web* concept receives lots of attention. The content hidden behind web forms which is invisible or hidden to general search engines like Google or Yahoo is defined as deep web *[16,24]* (also known as *hidden* or *invisible web* [14]). The growing understanding of deep

web and its potential enabling power have created a high demand from different businesses to get their hands on this huge source of valuable data. Whether the goal of a deep web access approach is indexing more representative content of a website (this approach is referred as *Surfacing approach* or *crawl-and-index* technique [33]) or extracting the whole content in deep websites, harvesters have a crucial role. Covering different domains, websites, and features enhances the need to have a general-purpose harvester which can be applied to different settings and situations. To develop such a harvester, a number of issues should be considered. Among these issues, business domain, targeted websites, and the harvesting goals are the most influential ones. Different business domains and goals could pose diverse characteristics on deep web access approaches. In some domains, a few number of big databases are the main sources of data and in others, data is scattered through a large number of websites. Facing a large number of websites makes it more desirable to have an approach with no need of extra configuration effort or at least with minimal configuration effort for each website. The goal of the harvesting task is also important [24]. In measuring the harvesting task performance, if the goal is to extract all the information in a website and the harvester downloads the data partially, the harvesting task is not considered successful. However, this might be a success story if the goal is just to obtain a representative set of data [24]. In addition to the domain and harvesting goal, features of deep websites could have great impacts on design and implementation of a deep web access approach. Different website features, from graphical interface to back-end designing and developing techniques, could play an important role. If a website is written in Flash [11], it is designed as an Applet, or it is a simple HTML page, it makes a big difference on the design of an access approach. The indexing policies, security issues, programming languages, dynamic or static nature of the pages, and a longer list of elements could affect the design of a harvester. Without a well-defined list of elements affecting harvesting tasks, especially the features of deep websites, having a general deep web access approach seems far from reach.

*Harvestability Factor Definition* To formalize all these mentioned important issues in accessing deep web data, a new factor is introduced in this paper as *Harvestability F*actor (HF). Although in a harvesting process, the roles of both harvester and website are intertwined, separate definitions are required by website and harvester designers for better understanding of harvesting processes. Hence, the HF is defined as an attribute of a website or a harvester representing the extent to which the website can be harvested ($HF_W$) or the harvester can harvest ($HF_H$).

As it is shown in Formula 1.1, the HF of a harvester is defined by applying a function $f$ on features of a given harvester and the goal of harvesting task. Each one of these harvester features represents the capability of the harvester in satisfying the corresponding challenges created by websites' features discussed in Section 3. All the harvester features have assigned weights represented by $w$. These weights define the importance of the related harvester feature. This importance is defined by the presence of the challenge this harvesting feature

deals with in the website or its frequency in a set of websites and the level of its impact on harvesting process. In this formula, general requirements which a harvester should meet are also included. These requirements are discussed in Section 4.

$$HF_H(Harvester, Goal) = f(w_1.harvester_{feature1}...w_N.harvester_{featureN}, harvesting_{goal}, harvester_{generalfeatures}) \quad (1.1)$$

Focusing on the role of websites in harvesting process, in Formula 1.2, the $HF_W$ is defined for a website by considering its features discussed in Section 3 and the given goal for the harvesting task. Each one of website features is assigned with a weight represented by $k$ in this formula. This weight represents the capability of a specific harvester or harvesters in general in satisfying the challenge created by this feature in harvesting task. This satisfaction is judged by the quality and amount of data harvested by harvester. Thinking about harvesting huge websites (like Google) supports the idea of having the website's size as another website feature in this formula. The size of a website poses different challenges on the harvester.

$$HF_W(website, Goal) = f(k_1.website_{feature1}...k_N.website_{featureN}, harvesting_{goal}, website_{size}) \quad (1.2)$$

Assigning values to the weights and features mentioned in these two formulas is beyond the scope of this paper and considered as a feature work. In this paper, it is tried to cover all aspects of the introduced HF elements; business domain, harvesting goal, harvester features and websites features to give a thorough guideline for designing general-purpose harvesters. Having considered current developments for deep web access approaches, all website features which have effect on harvestability of websites are studied. Without knowing about all the differences among websites and without studying their effects on the performances of harvesters, reaching a general-purpose and scalable harvester seems like a blind attempt. Having all these features in one big picture is vital for designing a general purpose harvester. As a reference business domain for test purposes, we focus on domain of job vacancies. In this domain, the focus is on a large number of small vacancy sites rather than a few large websites.

*Contributions* As one of the main contributions of this paper, a new concept, called Harvestability Factor (HF) is introduced. Through this concept, we attempt to put all the important elements in harvesting deep websites in one big picture. In addition to enabling websites' or harvesters' designers of evaluating where they products stand in harvesting point of view, the HF defines a framework for designing general deep web harvesters. In this framework, all the influential features from different aspects in designing a deep web harvester are covered. Some of these factors are mentioned in literature and the others are

discovered through the experiments by the authors. In literature, for designing harvesters, different approaches are introduced focusing on improving the general requirements of harvesters like performance, scalability and etc. We believe there is a gap in designing general deep web harvesters which roots from ignoring the features of deep websites. Without filling this gap, meeting the general requirements seems far from reach. Therefore, in this paper, having studied the current harvesters, we focus on the detailed features of deep websites which we find important in the design of harvesters. Based on these features, different categories are introduced which could be applied in addressing the capabilities of harvesters.

*Sections* In Section 2, different categories of developed harvesters in literature are mentioned to give an overview of current approaches applied by harvesters. Section 3 introduces the categories of deep websites based on the HF elements. In this section, all the detailed features of deep websites which have an effect on harvesting process are introduced and deep websites are categorized based on them. In Section 4, all the requirements for designing a general purpose deep web harvester from general requirements to detailed ones are discussed. Having mentioned all the necessary requirements, in Section 5, as a sample of such a general deep web harvester, the designed harvester by the authors of this paper is introduced. Finally, in Section 6, the conclusions drawn from this work are discussed and future work is suggested.

## 2 Categories of Harvesters

To access data behind web forms, a wide range of harvesters are suggested in literature [[35,39,36,31,30,41,42,27,23,10,25,13,19,14,24,33]]. The differences among these harvesters root from different sources; from applied techniques in each step of harvesting process to the main goal behind the harvester design. In this paper, the focus in categorizing harvesters is on the techniques and tools applied by harvesters to identify the data of interest. In the following, this classification is represented [30] [1].

1. HTML-based harvesters
   In HTML-based harvesters, the harvester relies on a set of different features of document HTML code [18,1,38]. To analyze the HTML structure of the pages, the document is translated into a parsing tree. This could be done by using browser controls like Internet Explorer to parse webpages into Data Object Model (DOM) trees. Then, by running a number of pre-defined extraction rules on the tree, the data is extracted.

[1] This categorization is introduced in [30] except number 6 and 7 which are added by authors of this paper. The category "harvesters based on wrapper development languages" [12,17,22] is also removed from this categorization. These harvesters are based on the other introduced categories. In such harvesters, new languages are introduced based on existing declarative languages to assist users in constructing wrappers in harvesters.

2. Harvesters based on Natural Language Processing (NLP) techniques
   In these harvesters [21,34,40], NLP techniques such as filtering, part-of-speech tagging, and lexical semantic tagging are applied to build relationships between phrases and sentences. From these extracted relationships, a number of extraction rules can be derived. These rules are based on syntactic and semantic constraints and help to identify the relevant information within a document.
3. Machine learning based harvesters
   These harvesters [26,29,15] rely on a given set of training examples to derive a number of extraction rules. In these techniques, rather than relying on linguistic constraints found in the page, rules are based on features of the structure of the pieces of data found.
4. Modeling-based harvesters
   In modeling-based harvesters [9,37], a data model is defined. In this data model, a number of objects, their properties and relationships are defined. Based on this data model and its modeling primitives, points of interest are located in Web pages.
5. Ontology-based harvesters
   In these harvesters [20], the extraction process is based on the data and not the presentation structure. These harvesters need a specific domain ontology. Through domain ontologies, concepts relevant to a particular topic or area of interest are defined and available for harvesters. The ontology-based harvesters use these ontologies to locate ontology's constants present in the page and to construct objects associated with them.
6. Computer vision based harvester
   These harvesters use computer vision techniques in addition to techniques from machine learning to analyze webpages. In these harvesters, the main goal is to identify and extract information from web pages by interpreting them visually as a human being does [2]. Some of these approaches use also the visual features on the deep Web pages [32].
7. Harvesters based on a combination of different tools introduced in previous categories. For example, in a harvester based on HTML structure, applying machine learning techniques could help in having more precise extraction results.

## 3 Deep Websites Categories

In this section, different features of deep websites which could be related to harvesting processes are studied. The roles of each one of these features as defining elements of website HF are also mentioned. Each of these features could be applied for categorizing deep websites.

### 3.1 Web Development Techniques

A number of techniques applied in developing and designing web sites and web pages create challenges for harvesters. These techniques are usually applied to

add interactivity to web pages as well as for improving site navigation. Also, some of the tools used for visually building web sites, generate pages which have scripting code. In following, there is a list of such techniques.

**Embedded scripting languages in HTML pages** This can be troublesome in client-side page content generation. Based on a user action, or change in a state, the content in layers are either shown or hidden. This could also be used for dynamically building HTTP requests for filling out a form and its submission. Managing HTML layers, performing redirections, dynamically generating navigations like pop-up menus and creating hidden anchors are some of the issues which could be handled by client-side scripts [11,3].

This technique actually prevents harvester to have the page as it is represented to user. Harvesters need to run the simulators or execution environments for the scripts embedded in these pages to become capable of having the same result as a user has. However, this is not the only problem. The content of the page could change based on users actions. Therefore, harvesters need to prepare for such situations too.

**Session management mechanisms** In session management mechanism [5], server keeps track of transactions made with a client. Based on this history and information on client resources, server could provide different services to the client.

Session management mechanism creates problems for current harvesters. Each session might include information about the user, its browser or other user-specific information [11]. In later access to documents and also distributed crawling, this will create problems for harvesters. Harvester needs to access all the session information it needs (such as cookies or the context for executing the scripting code). Also, in later access, the session can expire.

**Complex URL Redirections** For a list of very different purposes, from resolving similar or moved domains to manipulating search engines or visitors or even URL shortening, URLs are redirected. This means different responses are given to the browser request which results in browser showing a different page. These redirections could happen automatically or manually. Users could be asked to follow a link shown on the page or redirections could be done automatically by server scripts or scripts embedded in content of the pages [11].

It seems much easier for harvesters to deal with redirections handled on server side unless it is a redirection loop which does not load any page at the end or it is a redirect chain which might take longer time to have the final page returned.

Handling the redirections initiated by scripts embedded in page content is a completely different story for harvesters. Refresh meta tag in HTML, JavaScript redirections, and Frame redirections are examples of performing redirections by scripts in pages. In Frame redirections, the browser displays the URL of the frame document and not the URL of the target page in the URL bar. A JavaScript

redirection script could change the content of the page based on a user action or after a predefined time. These issues could result in different pages shown to users.

**Applets or Flash code** There are three different situations created by applying Flash or Applet in developing pages for harvesters. In the first situation, Flash or Applet is used for designing whole website. This makes it almost impossible for harvesters to access content without running expensive analysis over each item. In second situation, these technologies are used as the welcoming page. This could be easier for harvesters to deal with if this is recognized by them. The third situation is applying these techniques for parts of web pages which are not of great interest for users like advertisements. This parts could be removed and ignored. It is important to mention that websites designers avoid those practices in order to make sure their sites are on good terms with the crawler.

**Frames** There are also some issues such as frames which can create difficulties for harvesting processes. Detecting the right frame which contains the page content in a multi-frame page is one of the problems created by such issues.

**HTML Coding Practices** As mentioned before, HTML code of a page becomes highly important for harvesters which rely on data items tags, their attributes, and also presentation features like the size and place of the items. Therefore, knowing about the practices followed in writing these codes could be of great help for harvesters. There are a number of problems which might be faced by harvesters due to different HTML practices followed by a website. In following, a list of some of these problems is represented:

1. Having bad-written HTML code (like not closed tags) might cause problems in analyzing page HTML tree and therefore incapability of harvester to extract data.
2. As another coding practice, having ID, class, and other explanatory attributes for each item in page could be greatly helpful for harvesters. The lack of this data could also make it really hard for harvesters and make them prone to mistakes.
3. Having one HTML coding practice for all pages and data items. A simple logic behind these practices could also help. For example, if there is IDs for items, it should be the case for all of them or at least a well-defined set of the items (for different categories of data).
4. In some cases, data from the same category, even with the same presentation template have small differences in the HTML code behind them. This might mislead harvesters. For example, title of an item in most of the cases has hyper-link. However, in some cases it might be without any links. If harvester query does not include these small changes, it returns empty-handed as there is no <a> tag.

### 3.2 Website Policies

#### Search Policies

*Forms - Query Interfaces* To access data in a deep website, the first step is to know about the entrance point. There are a number of different web interfaces used currently in web sites. These interfaces could be classified in following categories:

- keyword-based search interfaces,
- form-like search,
- browsing search interface, and
- a combination of these interfaces [42].

Each one of these interfaces creates a different set of requirements for harvester. For example, in a form-like search interface, information on attribute-value bindings or accessing predefined lists of values for attributes could be of great help for harvesters to decide on which queries to send to the search engine. Harvesters should be able to find these interfaces on the websites. They should also be capable of distinguishing HTML forms for login, subscription, registration, polling, and message posting from query interfaces. Also, they should exclude "site search" which many web sites now provide. Recognizing other features of web forms could be also helpful. Knowing that query interface provides different search options like searching by keyword, industry domain, region, or time of publishing, can help harvester to act more efficiently.

*Indexing Policies* In the case of having the search box in a website, it becomes important to know about the indexed part of data. For example, with stop-words indexed in a web site, sending a stop-word query is one of the most reliable options to make sure there is a response to your query. Also, if there is no limitation on browsing through search results, sending only one stop-word could result in a high percentage coverage of website. Knowing about the indexing policy regarding the stop-words could help in defining your query generation methods. However, different defined sets of stop-words by different websites should be also considered.

In addition to indexing policies regarding stop words, it is of a great help for defining query generation mechanisms if harvesters know about which parts of data represented to users are indexed. For example, having only titles indexed will make great difference in defining next queries with having whole text of detailed pages indexed. This is the case in generating queries based on most frequent words in visited pages.

*Search Queries and Algorithms* In response to a query posed to a search engine, websites do not necessarily follow the same principles. In some cases, stop-words are removed from search queries, query phrases are treated in different ways (considered as AND phrase or OR phrase), or number of returned results shown to user is limited. There might be even differences on additional information

provided in reply to a query, such as statistics on search results and number of found related answers. There are also websites which put a limitation on the number of queries a client can send.

*Website Navigation (what is returned for search queries)* In most of the deep websites, a query is sent, search results are displayed and by following each one of those returned results, a detailed page is presented. However, there are cases in which this is not the case. Having posed a query on the website, accessing the detailed page is not so straightforward. In some websites, in return to a query, a list of categories related to that query are displayed. Following each one of those categories might end up in another subcategory. This makes it difficult for harvester to realize which returned page is a category or actually a detailed page.

**Security, Privacy and Legal Policies** As one of the most important issues which should be considered in a harvesting process is to check for privacy issues. Answering this question should be one of the first steps in harvesting process: "is it legal to access the data, store it and present it to users?". It is also important to note that if login information (having user-name and password) is required by website to access data. Considering website's terms of service to follow the privacy policies is also important.

In some websites, the *Robots Exclusion Protocol* is applied. Through this protocol, Web site owners give instructions about their site to web robots in a file named *Robots.txt*. In case of existence of such a file and depending on how strict it is asked to be followed, necessary concerns should be considered by crawlers and scrapers. Not all the websites welcome bots (harvesters, or crawlers) with open arms. Having recognized bots through traffic monitoring, bot identity declaration, or real person declaration techniques like a CAPTCHA, websites can use various measures to stop or slow them. Blocking an IP address, disabling web service API, commercial anti-bot services, or using application firewalls are some of these measures. It is also important to note other privacy policies of the website like policy on disclosing aggregate information for analytical purposes by owners of website.

### 3.3 Data and Content

**Type of Residing Data in Data Sources** The content of a deep website could be categorized in two different groups [24]:

- structured data like data in almost all shopping websites (products as entities), movie sites, job listings, and etc, and
- unstructured data like articles and papers. As an example of this category, Pub Med website [4] and Wikipedia [8] could be mentioned.

Each of these mentioned data types have different features which could be helpful in harvesters performances. For example, in a website representing structured

data, using the features of a data item like company name could help in defining next queries in crawling process resulting in a more efficient crawl. It is also of a great importance for harvesters to know about different data file formats for pdf, image, or video files. Different data formats need different handlers to download them.

**Data Presentation / Layout** How data is represented in web pages affects the harvesters relying on presentation-related features of data. Different data types in a website could be presented in different ways. For example, a website including data items of books, articles, shoes, and cloths could have similar or different templates for each of these items.

Even data items of a same category could be presented in different ways based on a set of different elements. For example, for data items from shoes category, different presentation templates could be applied based on features of the shoes like the shoes producing company. As another example, consider a game in a games website, with one platform and one score for that platform. In this case, the platform and score information is not presented as a table. If that game has several platforms, it has scores for each of those platforms. In this case, it is represented as a table. If these differences in presentation are not known to harvester, it will use the same algorithm to extract all data. This might result in extracting none or undesired information.

Structural variations on data presentation must be also tolerated by harvesters and treated accordingly. If the data is represented in a structured way like lists or tables or it is represented in text or a combination of both, harvester should treat them differently. At some point, it becomes also important to know if a data item has fields represented as nested data on pages; for example, "comments" or "scores information" which are usually a bigger item composed of a number of data items. This might pose different requirements on extracting and storage of information.

**Data Type Formats** Including ontologies and text-patterns in the process of extracting data from detailed pages makes it important to investigate how they can affect the harvesting process. Committing to one ontology and following same patterns for same concepts like "dd-mm-yyyy" format for all dates mentioned on the website could affect the configuration and design of the harvester. Also, for example, if the mentioned address format on the website is the same for all addresses mentioned in the pages of the website, it can have a great effect on the harvester configuration.

**Information of a Data Item is Scattered in Different Pages** Usually, the queries are sent to search engine, returned results are followed and data about desired items is extracted. However, this is not always the case. In some cases data of a interesting data item is scattered in website. In a more common way, general data is presented in the page navigated through search results.

However, more detailed information is provided in some other links which is accessible (only) through this detailed page (you need to go to the detailed page and then browse through the tabs or links to access the information you want). Finding these links and extracting information from them could be a challenge for harvesters.

**Providing Semantic Annotations (meta data)** The pages may include meta data or semantic markups and annotations. The annotations might be embedded in the pages or organized into a semantic layer [7] stored and managed separately from the web pages. Data schema and instructions from this layer can be retrieved by harvesters before scraping the pages.

**Website Content Language** Dealing with the language of the targeted website is one of the abilities that the harvesters should have. Some of the approaches applied in harvesters are based on parsing the content of web pages like data patterns. Having this in mind, it should be noted that dealing with Chinese language needs different configurations than English or the Farsi languages. Having different languages in the targeted websites will cause difficulties for these harvesters.

## 4 A General Purpose Harvester

As mentioned before, in the Introduction Section of this paper, designing a deep web access approach is highly affected by business domains, their features, and the harvesting goals. In this section, the effects of such issues on designing and implementing a general-purpose harvester will be studied.

### 4.1 High level requirements

Different deep web access approaches have different goals. In general search engines like Google, the goal of access approach is indexing more representative content of a website (this approach is referred as *Surfacing approach* or *crawl-and-index* technique [33]). In such search engines, having a more representative content of the website is the goal rather than extracting the whole residing content in deep websites. On the other hand, in domains like job vacancies, the goal is extracting all the data residing in deep websites to be able to represent it to users in different forms and enabling a wide range of different queries on them. However, some other domains like travel agencies, try to provide the access to different deep data sources through web portals. These sites are also knowns as aggregators. They design a general form on top of a set of different forms and use techniques like transforming and translating forms to provide users with one access point to all data they are looking for. Each of these goals poses different requirements on deep web access approach design. Despite these differences, they will also have similarities in design and implementation of different parts of the

corresponding deep web access approach. In this paper, we focus on harvesting all the data residing in a number of distributed target websites. To reach this goal, we need a harvester which can obtain a subset of data from the Web. This harvester should be :

- Automatic or run with least possible amount of configuration,
- Scalable; to be applicable to a large number of websites,
- Independent; independence of business domain, technology, and etc,
- Be efficient; with the least possible number of queries, harvests the most possible amount of data,
- Be easy to use; configuration and run settings should be easy for users to perform, and
- Resilient to changes both on website content and presentation.

With these features, a harvester should be able to go through all the following steps:

1. Fill in forms efficiently and automatically [42]. It should be able to determine which input fields in the form need to be filled. Detecting bindings and correlations among the form inputs, and deciding on values of those input fields to have a more efficient crawling process are also requirements of a harvester.
2. Extract data/entities from the returned results pages [42]. The harvester should be able to
   (a) Walk through all the returned results pages, and
   (b) Extract data from each page.
3. Store the extracted data [42]. It should
   (a) Keep the structured format of data, and
   (b) Combine pieces from multiple web pages to complete information on an entity.

In [42], two more steps are also considered for a harvester; discovering deep web sources of interest, and presenting extracted data to users and providing them with posing query mechanisms.

For all these steps considered in a deep web access process, the harvester should have an automatic error/change detection. This will help the harvester to be capable of doing an uninterrupted harvest as it becomes capable of detecting and resolving issues like IP based blocking, website failures, and etc. The harvester should be capable of providing firm guaranties about the exhaustive coverage of the harvested part of the Web. Size estimation of deep websites [28] and also defining a stop condition for harvesting process could be helpful in reaching this goal. In monitoring entities on web, it becomes highly important if the harvester could be able to keep the data up-to-date. This needs harvester to be capable of detecting new entities and deleted entities on the Web.

While fulfilling these mentioned high level requirements, the harvester should be also capable of harvesting all different categories of websites mentioned in Section 3. Therefore, in the following section, how to meet these requirements is discussed.

### 4.2 Detailed Features of a General Harvester

For a harvester to become capable of harvesting all different categories of websites mentioned in Section 3, it seems necessary that harvesters and crawlers could provide a proper execution environment for rendering pages. In this execution environment, not only the rendering environments for different scripts embedded in web pages should be provided but also it should be assured that all the information needed by those scripts from user-specific data to browser-related specifications are provided.

Some of the issues mentioned in categorizing deep websites into Simple and Complex ones could be resolved by simulating the execution environment. However, there are still some issues which might be faced during a harvesting task which remain still unresolved. Issues like Pop-up menus, hidden anchors, user-action-related or time-related changes in page content or presentation, session management related issues, Applets, Flash codes, multi-frame pages, and complex URL redirections.

1. Resolving embedded scripting languages in HTML pages
   As mentioned in Section 3, some scripts, based on a user action, or change in state, can redirect the page, change its content, or generate navigations like pop-up menus dynamically. To resolve these issues, harvesters need to not only provide execution environments for the scripts embedded in these pages, they should also simulate a user experience on the site. All the different states created by these client-side scripts should be detected. It can also represent navigation sequence followed by system to reach document.
2. Resolve session management mechanisms
   To resolve session management related issues, harvester needs to keep track of transactions made between server and client. Having this history and information on client resources could help harvesters to be able to do distributed crawling, and also access documents even with an expired session. In [11], a new language is introduced to define sequences of events in a web browser (events like login, find a form with name, and assign values and click on buttons).
3. Complex URL redirections
   As mentioned earlier in Section 3, it seems much easier for harvesters to deal with redirections handled on server side unless it is a redirection loop which does not load any page at the end or it is a redirect chain which might take longer time to have the final page returned. Refresh meta tag in HTML, JavaScript redirections, and Frame redirections are examples of performing redirections by scripts in pages. It seems the best option to resolve this issue, is to keep track of all the states that the page goes through. Waiting for all the redirections to be performed could be also a solution. However, this might ignore redirections performed in frames or the ones by fired by user-action related events.
4. Applets or Flash code
   If Flash or Applet is used for designing whole website, it seems so time-consuming for harvesters to access its content. However, if these technolo-

gies are used as welcoming page, the links referred by these codes could be followed.

5. Detecting the right frame which contains the page content in a multi-frame page should be also resolved by harvesters. Keeping track of all the redirections and changes related to them in the page, in addition to following and simulating client-side scripts could be considered as solutions to this problem.
6. Website policies
   To be able to resolve the issues regarding different website policies in indexing, search, security and privacy areas, the harvester should be able to detect the policies first. Detecting these policies will allow the harvester to choose a proper approach to address that policy. A pre-defined set of solutions based of different scenarios could be helpful.
   Dealing with anti-bots policies should be also considered by harvesters. For example, IP blocking, CAPTCHA, Robots.txt and other measurements taken by website owners to block bots should be resolved or mitigated.

In addition to all these issues which need to be resolved, there are a number of techniques which can help the harvester to improve its performance. Detecting empty pages and resolving the deduplication of pages are two of those techniques. It can be also helpful for harvesters to apply techniques like entity identification, and entity deduplication. Detecting relations among entities could provide additional information which can be helpful for storage and representation phases of the harvesting process.

In next section, current attempts to provide such harvesters are represented.

## 5 Harvestability Factor Validation

As mentioned in the Introduction Section, the HF can be used for evaluation of the extent to which a harvester can harvest and a website can be harvested. It was also discussed that this factor can work as a framework. To validate these claims, a collection of deep websites is studied considering the HF elements. Through this study, it is shown how these websites are categorized by applying the HF elements and how this can guide the design and implementation of a harvester. Having studied the set of deep websites and prioritizing the features of deep websites, the developed harvester by authors of this paper, as an example effort for developing a general purpose deep web harvester, is applied on the test set. By applying this harvester on this set of websites, it is shown that how these features are effective on harvesting processes in practice.

### 5.1 Test Set

To create the test set for illustrating how deep websites can be categorized based on HF and how this can be used in designing a general purpose harvester, a set of deep websites from the list of top-100 job vacancy websites in Netherlands

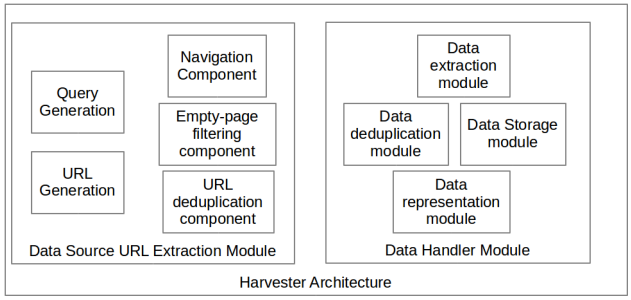


**Fig. 5.1.** Architecture of the developed harvester

is created [6]. For each of these websites, all the elements of HF is studied. To examine the harvester performance on each one of categories, the harvester is applied on the websites.

### 5.2 Developed Harvester

The developed harvester is a HTML-based harvester which automates loading of pages in a browser. These features help to resolve the challenges caused by some of the websites' features mentioned in Section 3. For example, to enable the harvester of implementing embedded scripts in HTML pages, the techniques for automating browsers are applied. Also, for selecting the points of interests, HTML-based techniques are considered. These features also help the harvester to meet some of the general requirements mentioned in Subsection 4.1 like automation, scalability, independency, efficiency, and being easy to use. For efficiency purposes, different query generation mechanism could be applied to have the most amount of data harvested with the least possible number of posed queries. The configuration is limited to entering the template, and XPaths for points of interests. There is also no need to enter a data model for data storage. Given these configurations for each website, high scalability level can be achieved. Domain-independency is also highly achieved through using only HTML-based techniques which also makes it language-independent.

The architecture of this harvester is shown in Figure 5.1. As shown in this figure, the harvester is composed of two main modules; one for generating URLs of data sources, and the other one for extracting, storing and representing extracted data. In this design, discovery of deep web sources and automatic form submissions are skipped and considered known to the harvester.

### 5.3 Results

Studying this set of websites from the domain of job vacancies brings a number of facts into light which will be explained in following. First, if we consider this set of websites big enough to represent the domain of job vacancies, the results can guide the design process by emphasizing on the elements of HF faced

more frequently among the websites of this domain. As it can be seen in Table 5.1, embedded scripts, detecting query interfaces, different data layouts, and in-persistent data patterns need further attention during the harvester design process.

Being based on browsers enables our harvester to overcome some of the challenges caused by embedded scripting languages in HTML pages. This is perfectly valid when there is no change of content based on user interaction with the page. However, having embedded scripts changing the content of the page based on user interaction or change of browser or spent time makes it more difficult for the harvester. Simulating user actions or changes in the page environment and comparing the generated result page with the previous version of the page should be performed in order to being capable of completely harvesting the page presented to users. This part is not included in our current version of harvester. However, it is worth mentioning that this type of scripts was not faced in our test collection. So, it was reasonable to postpone the implementation of this feature.

The second most common HF element in the test set is detecting query interfaces. In all the cases, our harvester could detect the template and query the search engines. The other common faced feature is having different data layouts. This is resolved in our harvester by making it possible to define different page templates for each website. However, this might pose a lot of time and effort during configuration phase if there are a large number of page templates used for showing data. In the HTML-based approaches, data can be extracted also based on the content. Therefore, if the data patterns are consistent, a high quality data extraction is still possible through the techniques applied by our harvester. Among the websites in the test set, 15 percent of the websites have limitation on browsing the number of viewed search results. To resolve this problem, different query generation mechanisms are applied which allow efficient harvesting of deep website. The harvester can also detect if stopwords are indexed or not and send the next queries accordingly. These meet two other common HF elements mentioned in Table 5.1.

Among the samples, it was observed that users are asked to enable the cookies for the website. This technique is becoming more frequently used by web developers. Therefore, harvesters should accordingly be able to recognize and resolve it. To resolve other session management techniques, keeping the session information and tracking the navigation path to the page are useful. In a not straight-forward search navigation website, which results in more steps than going through search, browsing results page, and viewing the detailed page, the developed harvester could work successfully. This was provided that there are only two types of page templates; search results page, and detailed page templates. The harvester can distinguish only these two types.

As it can be seen in Table 5.1, for some of the HF elements, no websites in the test set were found. This might be due to the specifications of the test domain. For example, the application of techniques like Applet or Flash could be seen more frequently in domains like Graphics or Music industries and not so often in job vacancy domain. The same applies to requiring credentials to

view job vacancies which is unacceptable in business models of these companies. It is also worth mentioning that defining some of these elements in HF for a website is time-consuming and sometimes hard. Persistent coding practices is one of those elements. It is time-consuming to study a website if it follows a persistent coding paradigm unless you face an exception. As it is shown in Table 5.1, for most of the elements, the harvester can perform successfully. However, to mediate the situations in which the harvester is not still able to resolve it completely, applying an error-detection approach in extracting data of interest by harvester and presenting it to user to reflect on that is a possible solution that is followed by our harvester.

## 6 Conclusions and Future Work

*Conclusion* As discussed in Section 5, the elements of the introduced harvestability factor can categorize deep websites based on their features which are important in harvesting process. This enables the owners of deep websites and website designers of evaluating where their products stand from harvesting point of view. This helps them to decide about which measures to take in order to follow their policies whether it is increasing access or limiting it.

For harvester designers, the harvestabiltiy factor acts not only as an evaluation metric of how well the harvester can behave in practice dealing with different websites, it also behaves as a framework for designing deep web harvesters. The HF provides designer with a thorough list of requirements they should meet and also helps to prioritize the features to be addressed and included in the harvester. Categorizing deep websites based on their harvestabiltiy factors makes it feasible to understand the importance of different websites' features. This helps to prioritize the features to be addressed and included in the harvester. The HF can also be applied for comparison of different deep web harvesters if it can be measured or also evaluating if a harvester can meet the needs for a general purpose deep web harvester.

*Future Work* Although having all the important features affecting the harvesting process is of a great value for designers, if the HF formulas can provide concrete values, applying them for comparison of harvesters or measuring success of a harvester could be more useful. Therefore, as one of the future work, our focus is on assigning values to the mentioned variables in the HF formulas. As another future work, we aim at using the HF in guiding us in developing a more general deep web harvester. Using the studies performed in this paper and extending them to a bigger test set will help us in deciding on the features our deep web harvester should include and prioritizing their developments.

## 7 Acknowledgement

We thank the WCC Company for hosting the first author and Jan de Vos, Eliska Went, and Marko Smiljanić for their support, discussions, and valuable input. This publication is supported by the Dutch national program COMMIT.

## References


1. Xwrap: An xml-enabled wrapper construction system for web information sources. In *Proceedings of the 16th International Conference on Data Engineering*, ICDE '00, pages 611–, Washington, DC, USA, 2000. IEEE Computer Society.
2. Diffbot - using computer vision to reinvent the semantic web. `http://www.xconomy.com/san-francisco/2012/07/25/diffbot-is-using-computer-vision-to-reinvent-the-semantic-web/`, 2013.
3. Dynamic web page. `http://en.wikipedia.org/wiki/Dynamic_web_page`, 2013.
4. Pubmed - us national library of medicine, national institutes of health. `http://www.ncbi.nlm.nih.gov/pubmed`, 2013.
5. session management. `http://en.wikipedia.org/wiki/Session_(computer_science)`, 2013.
6. The top 100 websites for your career. `http://www.forbes.com/sites/jacquelynsmith/2013/09/18/the-top-100-websites-for-your-career/`, 2013.
7. What is freeformat. `http://www.gooseeker.com/en/node/knowledgebase/freeformat`, 2013.
8. wiki-website. `http://www.wikipedia.org/`, 2013.
9. Brad Adelberg. Nodose - a tool for semi-automatically extracting structured and semistructured data from text documents. In *SIGMOD Record*, pages 283–294, 1998.
10. The New York Times Company Alex Wright. Exploring a deep web that google can not grasp. `http://www.nytimes.com/2009/02/23/technology/internet/23search.html?pagewanted=all`, 2012.
11. Manuel Alvarez, Alberto Pan, Juan Raposo, and Angel Vina. Client-side deep web data extraction. In *Proceedings of the E-Commerce Technology for Dynamic E-Business, IEEE International Conference*, CEC-EAST '04, pages 158–161, Washington, DC, USA, 2004. IEEE Computer Society.
12. Gustavo O. Arocena and Alberto O. Mendelzon. Weboql: Restructuring documents, databases, and webs. *Theor. Pract. Object Syst.*, 5(3):127–141, August 1999.
13. Luciano Barbosa and Juliana Freire. Siphoning hidden-web data through keyword-based interfaces. In *SBBD*, pages 309–321, 2004.
14. Michael Cafarella. Extracting and Querying a Comprehensive Web Database. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2009.
15. Mary Elaine Califf and Raymond J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pages 328–334, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.
16. BrightPlanet Corporation. Deep web intelligence. `http://www.brightplanet.com`, 2012.
17. Valter Crescenzi and Giansalvatore Mecca. Grammars have exceptions. *Inf. Syst.*, 23(9):539–565, December 1998.
18. Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 109–118, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

19. Anish Das Sarma, Xin Dong, and Alon Halevy. Bootstrapping pay-as-you-go data integration systems. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 861–874, New York, NY, USA, 2008. ACM.
20. D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, D. W. Lonsdale, Y.-K. Ng, and R. D. Smith. Conceptual-model-based data extraction from multiple-record web pages. *Data Knowl. Eng.*, 31(3):227–251, November 1999.
21. Dayne Freitag. Machine learning for information extraction in informal domains. *Mach. Learn.*, 39(2-3):169–202, May 2000.
22. Joachim Hammer, Héctor García-Molina, Svetlozar Nestorov, Ramana Yerneni, Marcus Breunig, and Vasilis Vassalos. Template-based wrappers in the tsimmis system. *SIGMOD Rec.*, 26(2):532–535, June 1997.
23. Hai He and Weiyi Meng. Wise-integrator: An automatic integrator of web search interfaces for e-commerce. In *In VLDB*, pages 357–368, 2003.
24. Yeye He, Dong Xin, Venky Ganti, Sriram Rajaraman, and Nirav Shah. Crawling deep web entity pages. Submitted to *VLDB 2012*.
25. Jer Lang Hong. Deep web data extraction. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 3420–3427, 2010.
26. Chun-Nan Hsu and Ming-Tzung Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Inf. Syst.*, 23(9):521–538, December 1998.
27. Lu Jiang, Zhaohui Wu, Qinghua Zheng, and Jun Liu. Learning deep web crawling with diverse features. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '09, pages 572–575, Washington, DC, USA, 2009. IEEE Computer Society.
28. Mohammadreza Khelghati, Djoerd Hiemstra, and Maurice van Keulen. Size estimation of non-cooperative data collections. In *Proceedings of the 14th International Conference on Information Integration and Web-based Applications & Services*, pages 239–246. ACM, 2012.
29. Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artif. Intell.*, 118(1-2):15–68, April 2000.
30. Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran S. da Silva, and Juliana S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Rec.*, 31(2):84–93, June 2002.
31. Jun Liu, Lu Jiang, Zhaohui Wu, and Qinghua Zheng. Deep web adaptive crawling based on minimum executable pattern. *J. Intell. Inf. Syst.*, 36(2):197–215, April 2011.
32. Wei Liu, Xiaofeng Meng, and Weiyi Meng. Vide: A vision-based approach for deep web data extraction. *IEEE Transactions on Knowledge and Data Engineering*, 22(3):447–460, 2010.
33. Jayant Madhavan, David Ko, Lucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's Deep Web crawl. *Proc. VLDB Endow.*, 1(2):1241–1252, August 2008.
34. Ion Muslea, Steven Minton, and Craig A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1-2):93–114, March 2001.
35. Alexandros Ntoulas, Petros Zerfos, and Junghoo Cho. Downloading textual hidden web content through keyword queries. In *Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, JCDL '05, pages 100–109, New York, NY, USA, 2005. ACM.

36. Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 129–138, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.
37. Berthier Ribeiro-Neto, Alberto H. F. Laender, and Altigran S. da Silva. Extracting semi-structured data through examples. In *Proceedings of the Eighth International Conference on Information and Knowledge Management*, CIKM '99, pages 94–101, New York, NY, USA, 1999. ACM.
38. Arnaud Sahuguet and Fabien Azavant. Building intelligent web applications using lightweight wrappers. *Data Knowl. Eng.*, 36(3):283–316, March 2001.
39. Cheng Sheng, Nan Zhang, Yufei Tao, and Xin Jin. Optimal algorithms for crawling a hidden database in the web. *Proc. VLDB Endow.*, 5(11):1112–1123, July 2012.
40. Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Mach. Learn.*, 34(1-3):233–272, February 1999.
41. Gerhard Weikum and Martin Theobald. From information to knowledge: harvesting entities and relationships from web sources. In Jan Paredaens and Dirk Van Gucht, editors, *PODS*, pages 65–76. ACM, 2010.
42. Nan Zhang and Gautam Das. Exploration of deep web repositories. *PVLDB*, 4(12):1506–1507, 2011.

**Table 5.1.** Test Set of Websites For HF Elements

| ***Harvestability Factor's Element*** | ***Percentage of Sample Websites Having the Element**** | ***The Harvester Performance*** |
|---|---|---|
| *Embedded Script in HTML 3.1* | *100 percent* | *Harvester was successful in dealing with this element for all the cases.* |
| *Applet / Flash 3.1* | *0 percent* | *This feature is not included. In case of facing this element, harvester fails.* |
| *Data Layout (different layouts) 3.3* | *26 percent* | *Harvester needs pre-configuration for different page templates.* |
| *Navigation (not straight-forward) 3.2* | *2 percent* | *Successful (can differentiate only BTW search result pages and detailed pages)* |
| *Muli-page data source 3.3* | *2 percent* | *Harvester needs pre-configuration.* |
| *Search Policies (limited search results) 3.2* | *14 percent* | *Using different query generation mechanisms resolves this situation* |
| *Indexing Policies (not stopwords) 3.2* | *10 percent* | *Harvester detects if stopwords are indexed or not and sends next queries accordingly* |
| *HTML Coding Practices (not persistent) 3.1* | *0 percent (all sample websites are persistent in coding)* | *(Not faced but could make problem as it is HTML based harvester)* |
| *Security / Privacy / Legal Policies 3.2* | *0 percent (no websites with username, pass, or limitation for bots)* | *Not faced but using browsers resolves bots limitations* |
| *URL Redirection 3.1* | *14 percent* | *Harvester needs pre-configuration to resolve this.* |
| *Residing Data (text, no structure) 3.3* | *10 percent* | If structured or semi-structured, if text, needs analyzing tools |
| *Session Management 3.1* | *2 percent* | Successful in dealing with cookies |
| *Query Interface Type 3.2* | *100 percent (all have text search or browsing features)* | Successful |
| *Persistent Data Patterns (not persistent) 3.3* | *24 percent* | Successful if the data layout is defined |
| *Multi-frames 3.1* | *0 percent (no website with framing issues)* | *(Not faced but could make problem)* |

* This column helps in recognizing which elements play a more important role in harvesting the targeted websites.